Fast Best-Effort Pattern Matching in Large Attributed Graphs

Hanghang Tong[†] Brian Gallagher[⋆] Christos Faloutsos[†] Tina Eliassi-Rad[⋆]
[†]Carnegie Mellon University [⋆]Lawrence Livermore National Laboratory
[†]{htong, christos}@cs.cmu.edu [⋆]{bgallagher, eliassirad1}@llnl.gov

ABSTRACT

We focus on large graphs where nodes have attributes, such as a social network where the nodes are labeled with each person's job title. In such a setting, we want to find subgraphs that match a user query pattern. For example, a 'star' query would be, "*find a CEO who has strong interactions with a Manager, a Lawyer, and an Accountant, or another structure close to that as possible*". Similarly, a 'loop' query could help spot a money laundering ring.

Traditional SQL-based methods, as well as more recent graph indexing methods, will return no answer when an exact match does not exist. This is the first main feature of our method. It can find exact-, as well as near-matches, and it will present them to the user in our proposed 'goodness' order. For example, our method tolerates indirect paths between, say, the 'CEO' and the 'Accountant' of the above sample query, when direct paths don't exist. Its second feature is scalability. In general, if the query has n_q nodes and the data graph has n nodes, the matching problem needs polynomial time complexity ($O(n^{n_q})$), which is prohibitive. Our *G-Ray* ("*Graph X-Ray*") method finds high-quality subgraphs in time linear on the size of the data graph.

Experimental results on the DLBP author-publication graph (with 356K nodes and 1.9M edges) illustrate both the effectiveness and scalability of our approach. The results agree with our intuition, and the speed is excellent. For instance, it takes 4 seconds on average for a 4-node query on the DBLP graph.

1. INTRODUCTION

Given a large graph with attributed nodes, how can we quickly find patterns that match, say, the 'star' query of the abstract? And what should we do when no exact instance of the specified pattern exists?

We propose *Graph X-Ray* (G-Ray), a fast method that finds subgraphs that either match the desirable query pattern exactly, or as well as possible. We propose an intuitive goodness score $g()$ to measure how well a subgraph matches the query pattern, and we give a fast algorithm to find and rank qualifying subgraphs. The idea of best-effort is illustrated by an example. Figure 2.a shows a 'line' query on the fictitious graph of Figure 1; since no instance of the query exists, our system returns a 'best-effort' match, as shown in Figure 2.b. Traditional SQL-based methods, as well as more recent graph indexing methods, will return no answer when an exact instance of a pattern does not exist.

Contributions. *G-Ray* provides a framework and a method for quickly finding the best-effort subgraphs that qualify for a given pattern query on large (categorically) attributed graphs, like author-conference networks (DBLP). Our main contributions are:

Effectiveness: G-Ray returns the best-effort results. That is, the matching subgraphs will include all the nodes in the pattern query and will conform to the pattern query's graph structure – even when the exact pattern does not exist in the data graph. The method carefully tolerates longer, indirect paths, as guided by our proposed goodness score $g()$.

Scalability: G-Ray scales up linearly (instead of polynomially) with respect to the size of the data graph.

The rest of the paper is organized as follows. Section 2 describes the formal definition of our inexact subgraph matching problem. Sections 3 and 4 provide the overview and details of our proposed approach, respectively. Our experimental results are in Section 5, and related work in Section 6. We conclude the paper in Section 7.

2. PROBLEM DEFINITION

Here, we give the formal problem definition. To start with, we assume that only nodes in the graph have categorical attributes. We shall use a running example of the fictitious social network of Figure 1, where nodes indicate people, the (weighted) edges indicate volume of communication (e.g., number of phone-calls exchanged), and the shape of each node indicates the job-title.

In this setting, the problem of Best-effort Subgraph Matching is defined as follows

PROBLEM 1. *Best-effort Subgraph Matching*

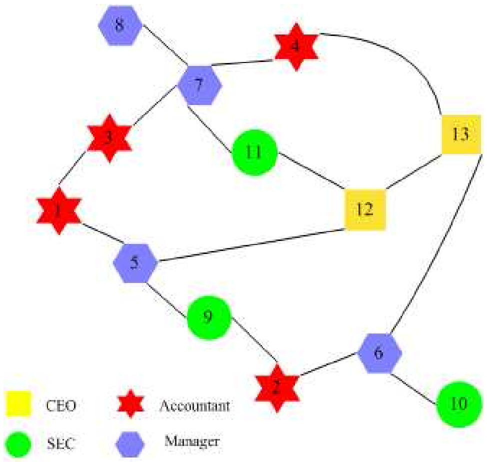


Figure 1: A simplified social network with attributes on nodes. 'CEO's (in Yellow Squares), 'SEC' (secretaries, in green circles), etc

Given: *(i) A (large) graph $\mathcal{G}$ whose nodes have one categorical attribute (like 'job-title'), (ii) a query (small) graph $\mathcal{H}_q$ showing the desirable configuration of professionals (e.g., a square-star-hexagon-circle loop, as in Figure 2.e), and (iii) the number of desired matching subgraphs n'.*

Find: *n' matching subgraphs $\mathcal{H}_t$ ($t = 1, \ldots, n'$), that match the query $\mathcal{H}_q$ as well as possible, according to a goodness score $g()$.*

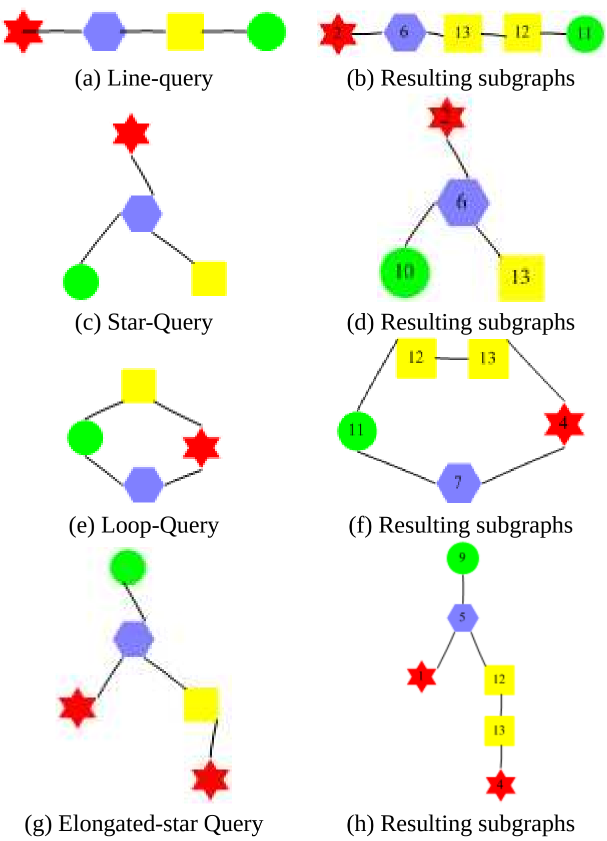
Next, we will define our goodness scoring function $g()$, after we define some preliminary, important terms. Notice that the graphs $\mathcal{H}_q$ and $\mathcal{H}_t$ are *qualitatively* different: the nodes of $\mathcal{H}_q$ are attribute values (e.g., 'CEO,' 'Lawyer,' etc), while the nodes of the subgraph $\mathcal{H}_t$ are data nodes (e.g., people like 'John Smith,' 'Jane Doe,' etc).

2.1 Terminology

We say that a subgraph $\mathcal{H}_t$ (as in Figure 2.f) conforms to a query graph $\mathcal{H}_q$ (say, as in Figure 2.e, if the subgraph has all the appropriate job-titles, with the correct connections between them, except that some connections may be indirect, including additional nodes. We shall refer to these extra nodes as intermediate nodes, and to this phenomenon as interception. The non-intermediate nodes will be referred to as matching nodes. Thus, node '12' is an intermediate node in Figure 2.f, because, without it, nodes 11-13-4-7 would form a perfect loop, matching the loop query of Figure 2.e. Similarly, node '13' can be viewed as an intermediate node in the same setting with nodes 11-12-4-7 forming a perfect loop.

Whenever there is a matching subgraph $\mathcal{H}_t$ we say that its matching nodes instantiate the corresponding nodes of the query graph $\mathcal{H}_q$, and also that the subgraph instantiates the query. In the example above (Figures 2.e-f), node '11' instantiates the circle node ('secretary') of the loop query graph.

2.2 Goodness Function



(a) Line-query

(b) Resulting subgraphs

(c) Star-Query

(d) Resulting subgraphs

(e) Loop-Query

(f) Resulting subgraphs

(g) Elongated-star Query

(h) Resulting subgraphs

Figure 2: Examples of queries and results by *G-Ray*

How can we measure the goodness of a match $g()$ between a (conforming) subgraph $\mathcal{H}_t$, and a query graph $\mathcal{H}_q$? Intuitively, if two nodes are adjacent in the query graph $\mathcal{H}_q$, their matching nodes should have good 'proximity' in the matching subgraph $\mathcal{H}_t$. There are two questions: (a) how to measure the proximity of two nodes in a graph and (b) how to combine all these proximity scores.

For the first question, we propose to measure the proximity $r_{i,j}$ between node i and node j as the score of j on a random walk with restarts, when node i is the restarting node. Once we decide the fly-out probability c (e.g. $c = 0.1$, as was used in [20]), all the $r_{i,j}$ scores are well defined, between any two nodes in our data graph $\mathcal{G}$.

For the second question, we propose to consider only the edges of the query graph, and aggregate the proximity scores $r_{i,j}$ of all the pairs of (i, j) matching nodes, where nodes i and j match nodes of the query graph that are adjacent. For example, in the query and subgraph example of Figures 2.e-f, and treating node '13' as the intermediate node, the goodness score would be the combination of scores going clockwise on the edges $r_{11,12}$, $r_{12,4}$, $r_{4,7}$, $r_{7,11}$, and counter-clockwise: $r_{12,11}$, $r_{11,7}$, $r_{7,4}$, $r_{4,12}$. How should we combine these scores? Should we add them? or consider them in triplets of nodes ('chains')? or in some other way? It turns out that we can take their product, which has a probabilistic interpretation. It is the probability that the appropriate random particles, walking on the full data graph $\mathcal{G}$ with restarts, will find themselves on the matching nodes of the subgraph $\mathcal{H}_t$.

Mathematically, we define the goodness score $g(\mathcal{H}_q, \mathcal{H}_t)$ of a subgraph $\mathcal{H}_t$ with respect to a query graph $\mathcal{H}_q$, as the product of $r_{i,j}$ proximity scores of the matching nodes, taken pairwise according to the matched edges of $\mathcal{H}_q$.

DEFINITION 1 (GOODNESS FUNCTION). *Consider a query graph $\mathcal{H}_q$ and a conforming data subgraph $\mathcal{H}_t$, with matching function $m(i) = v$ (i.e., data node i matches/instantiates query node v), then the goodness function $g(\mathcal{H}_q, \mathcal{H}_t)$ is defined as*

$$g(\mathcal{H}_q, \mathcal{H}_t) = \prod_{i,j} r_{i,j} \quad \text{where } ((m(i), m(j)) : \text{edge in } \mathcal{H}_q \tag{1}$$

Thus, Problem 1 is well defined. Given a data graph $\mathcal{G}$ and a query graph $\mathcal{H}_q$, find the best n' matching subgraphs (best according to the aforementioned goodness function $g()$).

2.3 Discussion

Problem 1 is polynomial for fixed-size pattern queries. This is prohibitive for large data graphs. Suppose you have a data graph $\mathcal{G}$ with size $n = (|V|)$ and a query graph $\mathcal{H}_q$ with size $n_q = (|V_q|)$, then for a **fixed-size** n_q the subgraph isomorphism problem is polynomial($O(n^{n_q})$). *G-Ray*, on the other hand, has time complexity linearly on the size of the data graph.

There are some additional observations and potential generalizations, before we go to an example. In this work, we assume there is only one attribute (eg., job-title), with m possible categorical values (v_1= 'CEO', v_2= 'Manager' etc, in our example). Formally, the attributed graph $\mathcal{G}$ can be described by an $n \times n$ node-to-node matrix $\mathbf{W}$ and an $n \times m$ node-to-attribute matrix $\mathbf{A}$: $\mathcal{G} = \{\mathbf{W} = [w_{i,j}], \mathbf{A} = [a_{i,k}]\}$. Every two nodes (i, j) are associated with a nonzero weight $w_{i,j}$ if there exists an edge between them. For every node i, it is associated with an attribute vector $\vec{a}_i = [a_{i,1}, ..., a_{i,m}]^T$: $a_{i,k} = 1$ if node i is labelled with k^{th} attribute value; 0 otherwise.

The query $\mathcal{H}_q$ is another (usually much smaller compared with $\mathcal{G}$) graph. The nodes of $\mathcal{H}_q$ are labelled with 1-out-of-m attribute values, indicating what kinds of nodes we want, while the edges of $\mathcal{H}_q$ indicate what kinds of connection we require between different nodes. As $\mathcal{G}$, the query graph can also be denoted by two matrices: as $\mathcal{H}_q = \{\mathbf{W}_q, \mathbf{A}_q\}$. Similarly, every resulting subgraph is also denoted by two matrices: $\mathcal{H}_t = \{\mathbf{W}_t, \mathbf{A}_t\}$.

Table (1) gives all the symbols used in the paper. Following standard notation, we use calligraphic for subgraphs (e.g. $\mathcal{H}_q$, $\mathcal{G}, \mathcal{H}$), bold capitals for matrices (e.g.$\mathbf{W}$, $\mathbf{A}$), and an arrow for column vector (e.g. $\vec{a}_i$). Since we have two graphs ($\mathcal{G}, \mathcal{H}_q$) as inputs, for clarification, we reserve i, j as the indices for the nodes in $\mathcal{G}$, (i, j) as the index for the edges in $\mathcal{G}$. We reserve k, l as the indices for the nodes in $\mathcal{H}_q$, (k, l) as the index for the edges in $\mathcal{H}_q$. For the node i in $\mathcal{G}$, it can be uniquely identified by $\langle i, \vec{a}_i^T \rangle$. If the node i in $\mathcal{G}$ only has one attribute value k (or we only care for its k^{th} attribute value), we also denote it as $\langle i, k \rangle$ for simplification.

2.4 An Illustrative Example

As we mentioned, we allow best-effort matching, in the sense that we allow for indirect paths, when the desirable direct paths do not exist.

Figure 1 gives a simplified social network (who-talks-to-whom) with their job title as the node attribute, which can take 1-out-of-4 values:"Accountant", "Manager", "CEO", and "SEC" (short for 'secretary'). Thus, the who-talks-to-whom graph $\mathcal{G}$ is a 14 node-to-node matrix $\mathbf{W}$ and a 14×4 node-to-attribute matrix $\mathbf{A}$. For example, if we store the attribute values "Accountant", "Manager", "CEO", and "SEC" sequentially, the attribute vector $\vec{a}_4 = [1\ 0\ 0\ 0]^T$ since node 4 is labelled as "Accountant"(the first attribute value). Thus, we can identify node 4 in this graph by either $\langle 4, [1\ 0\ 0\ 0]^T \rangle$ or simply as $\langle 4, 1 \rangle$ (since here every node is only labelled by one attribute value.)

Figure 2 shows some sample queries as well as the corresponding results. Figure 2.a is a line-query, that is "finding Accountant, Manager, SEC and CEO such that, the qualifying Manager has strong connection with CEO as well as Accountant; while the qualifying CEO has strong connection with Manager and SEC". Figure 2.b shows a best-effort match (the connection between node 11 and node 13 is indirect)[1]. Figure 2.d shows an exact match for a star-query(c), that is "find an Accountant, a Manager, a SEC and a CEO such that the qualifying Manager has strong connections to the other 3". Figures 2.e-h show some more complicated queries and corresponding results. Again, the results are not exact, but best-effort.

3. PROPOSED METHODS: OVERVIEW

3.1 Preliminaries: Interaction with SQL

If we only wanted exact matches, we could write SQL queries to identify any and all of the patterns in the left column of Figure 2. *G-Ray* has two distinct advantages: (a) it can allow for best-effort matches (tolerating longer, indirect paths, when direct paths do not exist) and (b) due to our proposed goodness function $g()$, it can rank the output and avoid flooding the user with a potentially huge number of near-unimportant matches.

On the other hand, our method can easily incorporate SQL, if necessary; that is we can always use our algorithm **together with**, rather than **'against'**, SQL-based methods. For example, if there exist many exact matching results, we can use SQL as a pre-processing step for finding all the results and then feed them to *G-Ray* to find a few 'best' ones, and/or to rank the results.

3.2 Preliminaries: Random Walks and CePS

Our *G-Ray* method uses two stepping stones: the random walk with restart idea [16, 20] and of the CenterPiece Subgraphs idea [19]. The former is necessary to estimate our proposed goodness function $g()$, as shown in equation (1). There are fast algorithms to compute or partially pre-compute the desirable *proximity scores* $r_{i,j}$ for every pair of nodes $(i,\ j)$. *G-Ray* is completely independent of how the proximity scores are computed, and thus it can easily take advantage of any fast method, as well as any faster method that may appear in the future.

The other stepping stone is the CenterPiece Subgraphs (CePS), which operate on a plain graph (no attributes on the nodes), and find the few most central ('CenterPiece') people that are

[1] For the query examples shown here, *G-Ray* also finds other exact matches, e.g. the subgraph containing nodes 1, 5, 11, 12 for the line-query. For clarity of exposition, we omit them.

Table 1: Symbols

Symbol	Description
$\mathcal{G} = \{\mathbf{W}, \mathbf{A}\}$	the attributed graph
$\mathbf{W} = [w_{i,j}]$	the $n \times n$ node-to-node matrix ($i, j = 1, ..., n$) for $\mathcal{G}$
$\mathbf{A} = [a_{i,k}]$	the $n \times m$ node-to-attribute matrix($i = 1, ..., n$, $k = 1, ..., m$) for $\mathcal{G}$
n	the total number of nodes in the attributed graph $\mathcal{G}$
m	the total number of attribute values
n_l	the total number of nodes in $\mathcal{G}$ having attribute value l
i, j	the indices for nodes in $\mathcal{G}$. Correspondingly, (i, j) is the index for the edges in $\mathcal{G}$
$\vec{a}_i$	the attribute vector for node i in $\mathcal{G}$. $\vec{a}_i = [a_{i,1}, ..., a_{i,m}]^T$
$\mathcal{H}_q = \{\mathbf{W}_q, \mathbf{A}_q\}$	the attributed query graph
n_q	the number of nodes in the query graph
k, l	the indices for nodes in $\mathcal{H}_q$. Correspondingly,(k, l) is the index for the edges in $\mathcal{H}_q$
$\mathcal{H}_t = \{\mathbf{W}_t, \mathbf{A}_t\}$	the resulting matching subgraphs ($t = 1, ..., n'$)
n'	the number of required subgraphs
c	the fly-out probability of random walk with restart
$r_{i,j}$	the *steady-state* probability that a particle will find itself at node j when it does random walk with restart from node i in $\mathcal{G}$
$r_{l,k}$	the *steady-state* probability that a particle will finally find itself at attribute node k when it does random walk with restart from attribute node l in $\mathcal{H}_q$.

well connected to the k given query nodes. For example, if 'Smith', 'Johnson' and 'Thompson' are data mining researchers, the query would be *who are the researchers that are most central to all three of them?* (where links are papers co-authored). CePS is able to quickly find such central/CenterPiece nodes, and we make heavy use of it.

3.3 The Outline of *G-Ray*

Since we allow inexact match, there might be two types of nodes in resulting conforming subgraphs: *matching* data nodes and *intermediate* data nodes: nodes to bridge two matching nodes when no direct connection exists between them.

Given a query graph $\mathcal{H}_q$, how to start looking for promising subgraphs $\mathcal{H}_t$, that is, data subgraphs that may have high goodness score $g()$?

Our idea is best illustrated with an example. This time we shall use the 'line' query of Figure 2(a). At the high level, we want to find good starting points (seed data nodes), like square (CEO) nodes surrounded by many circles (SEC) and many hexagonal (Manager) nodes. Say we find that node '13' is the most promising such CEO node. The measure for 'promise' will be firmly defined next - and in fact, it is the CenterPiece node of a carefully designed setting.

Once we have decided on a good 'seed', we want to expand to create a full, conforming subgraph. For the line query scenario above, *G-Ray* will choose the best neighboring node of the necessary type (say, 'SEC'), and then look for the best path to connect them. In our example, suppose that node '11' is the best neighboring node, and *G-Ray* has to go through node '12' to connect the CEO at '13' with the 'SEC' at '11'.

The algorithm continues until the seed node '11' is expanded to a full, conforming subgraph (if possible); by its construction, the resulting subgraph will have a high goodness score.

We can repeat with another seed node, until the user has all n' matching subgraphs that he requested.

Thus, there are three basic modules in *G-Ray*:

- **Seed-Finder** : It selects a desired attribute-value node from the query graph $\mathcal{H}_q$; and it finds a "very promising" matching data node with that attribute value according to $\mathcal{H}_q$ when $\mathcal{H}_t$ is empty
- **Neighbor-Expander** : It expands the seed node, by finding a "good" matching node with the desired attribute value according to $\mathcal{H}_q$ when $\mathcal{H}_t$ is partially built
- **Bridge** : It finds a "good" path to connect two matching data nodes if they are required to be connected according to $\mathcal{H}_q$

Algorithm 1 *G-Ray*

```
Require: The attributed graph $\mathcal{G}$, the query graph $\mathcal{H}_q$,and the
    number of resulting subgraphs $n'$.
    Output: The resulting subgraphs $\mathcal{H}_t(t = 1, ..., n')$
 1: for $t = 1 : n'$ do
 2:     initialization
 3:     find matching node $\langle i, k \rangle$ by Seed-Finder ;
 4:     add $\langle i, k \rangle$ to $\mathcal{H}_t$, and mark node $k$ in $\mathcal{H}_q$ as "touched"
 5:     repeat
 6:         pick up a "touched" node $k$ in $\mathcal{H}_q$
 7:         for each of $k$'s "un-processed" edges $(k, l)$ in $\mathcal{H}_q$ do
 8:             find matching node $\langle j, l \rangle$ by Neighbor-Expander
 9:             find a "best" path between $i$ and $j$ by Bridge
10:             add it to $\mathcal{H}_t$; mark edge $(k, l)$ as "processed"
11:         end for
12:         update the status of node $k$ and $l$ in $\mathcal{H}_q$
13:     until every node in $\mathcal{H}_q$ is marked as "processed"
14: end for
```

It can be seen that *G-Ray* generates resulting conforming subgraphs $\mathcal{H}_t(t = 1, ..., n')$ one by one. For each subgraph, it first sets (step 2) $\mathcal{H}_t$ to be NULL; every node k in $\mathcal{H}_q$ is marked as "un-processed"; and every edge (k, l) in $\mathcal{H}_q$ is marked as "un-processed". Then, *G-Ray* builds the subgraph $\mathcal{H}_t$ gradually, by the above three modules: **Seed-Finder** **Neighbor-Expander** and **Bridge** . In additional, we also need to keep track of the status of the nodes and edges in the query graph $\mathcal{H}_q$, which is defined as following:

- An edge (k, l) in $\mathcal{H}_q$ is "processed" iff 1) there exist two matching nodes in $\langle i, k\rangle$ and $\langle j, l\rangle$ in $\mathcal{H}_t$, and 2) **Bridge** has been applied to these two nodes; otherwise the edge (k, l) is "un-processed".
- A node k in $\mathcal{H}_q$ is "processed" iff all of its adjacent edges have been marked as "processed"; the node k in $\mathcal{H}_q$ is "un-touched" iff all of its adjacent edges in $\mathcal{H}_q$ have been marked as "un-processed"; otherwise the node k in $\mathcal{H}_q$ is "touched"

4. PROPOSED METHODS: DETAILS

In this section, we provide the details of *G-Ray*. There are three basic modules of *G-Ray*, as we mentioned before. In the first two, **Seed-Finder** and **Neighbor-Expander** , we find those matching nodes with desired attribute values; the **Bridge** module identifies intermediate nodes (if necessary) and finds a "best path" to connect two matching nodes.

4.1 Seed-Finder

Seed-Finder takes the attributed graph $\mathcal{G}$, the query graph $\mathcal{H}_q$ and the one attribute value k in $\mathcal{H}_q$[2] as input, and outputs a qualifying seed node $\langle i, k\rangle$ in $\mathcal{G}$.

Let $g(\mathcal{H}_q, i)$ be the goodness function for a given node $\langle i, k\rangle$:

$$g(\mathcal{H}_q, i) \triangleq \prod_{j, j\neq i} r_{j,i} \quad (m(i) = k, m(j) = l) : \text{edge in } \mathcal{H}_q \tag{2}$$

It can be seen that $g(\mathcal{H}_q, i)$ is the contribution of node $\langle i, k\rangle$ for the total goodness function in equation (1). Thus, if all of $k's$ neighbors have been instantiated/matched, we can just choose seed node $\langle i, k\rangle$ by optimizing equation (2).

However, since the resulting subgraph $\mathcal{H}_t$ is empty, to ensure that the final subgraph $\mathcal{H}_t$ is well connected, a matching node $\langle i, k\rangle$ should also have high proximity score with **some unknown** node $\langle j, l\rangle$, even if the attribute value k is not directly adjacent to l in the query graph $\mathcal{H}_q$ (as long as they are closely related to each other), Moreover, if in the query graph $\mathcal{H}_q$, the attribute value k is closely related to two different attribute values l and l', we should give more weight to the attribute value that is more relevant with k. Finally, since the resulting subgraph $\mathcal{H}_t$ is empty, we really do not know which node $\langle j, l\rangle$ in graph $\mathcal{G}$ should be referred to. Thus, we relax this quantity to the average proximity score for node $\langle i, k\rangle$ wrt all the nodes $\langle j, l\rangle$ in graph $\mathcal{G}$.

Formally, $g(\mathcal{H}_q, i)$ in **Seed-Finder** is relaxed as follows:

$$g(\mathcal{H}_q, i) = \prod_{l,\ l\neq k} \Big(\frac{1}{n_l} \sum_{\{j|m(j)=l\}} r_{j,i}\Big)^{\frac{1}{r_{l,k}}} \tag{3}$$

where n_l is the total number of nodes in $\mathcal{G}$ having attribute l; and $r_{l,k}$ measures the proximity between l and k by random walk with restart on $\mathcal{H}_q$(see Table 1).

The pseudo code of **Seed-Finder** is given in Alg.(2). Note that in step 7, we maintain a global seed list (sl) which contains all the seeds found in the previous subgraphs($\mathcal{H}_1, ..., \mathcal{H}_{t-1}$). In this way, we ensure that different subgraphs have different seeds.

Algorithm 2 Seed-Finder

```
Require: The attributed graph $\mathcal{G}$, the query graph $\mathcal{H}_q$, and
    one attribute value $k$ in $\mathcal{H}_q$.
    Output: One matching seed node $\langle i, k\rangle$ in $\mathcal{G}$
1: for each $l \in \mathcal{H}_q (l \neq k)$ do
2:     compute $r_{l,k}$
3: end for
4: for each $\langle i, k\rangle$ in $\mathcal{G}$ do
5:     compute $g(\mathcal{H}_q, i)$ by equation (3)
6: end for
7: return: $i = \text{argmax}_{j\notin \text{sl}}\ g(\mathcal{H}_q, j)$
```

4.2 Neighbor-Expander

Neighbor-Expander takes as input the attributed graph $\mathcal{G}$, the query graph $\mathcal{H}_q$, one "touched" attribute value k in $\mathcal{H}_q$, and the partially built subgraph $\mathcal{H}_t$; and outputs a matching node $\langle i, k\rangle$ in $\mathcal{G}$.

The basic idea of **Neighbor-Expander** is similar with that of **Seed-Finder** . However, at this point, we already have the partially built subgraph $\mathcal{H}_t$, which makes the following difference between the two modules.

First of all, since k is marked as "touched", at least some of its edges in $\mathcal{H}_q$ must have been marked as "processed". Suppose edge (k, l) is marked as "processed", there must exist some matching nodes $\langle j, l\rangle$, which can be used in calculating $g(\mathcal{H}_q, i)$. Secondly, given a node $\langle i, k\rangle$, while **Seed-Finder** relaxes its goodness function $g(\mathcal{H}_q, i)$ to all attribute nodes (except node k itself) in the query graph $\mathcal{H}_q$; in **Neighbor-Expander** , since the resulting subgraph $\mathcal{H}_t$ is already partially built, we do not need this relaxation to ensure that the final $\mathcal{H}_t$ is well-connected. Finally, while in **Seed-Finder** , the (relaxed) average score (e.g. equation (3)) is weighed by the proximity between l and k, in **Neighbor-Expander** , it is not weighted (since every l is directly adjacent to k, $r_{l,k}$ does not make much difference).

Formally, the goodness function $g(\mathcal{H}_q, i)$ in this case is relaxed as equation (4). Note that the indicator function $I(l, k) = 1$ if edge (l, k) in $\mathcal{H}_q$ is marked as"processed", and 0 otherwise. Also the whole product is taken among k's directly adjacent neighbors in $\mathcal{H}_q$. The pseudo code of **Neighbor-Expander** is given in Alg. (3).

$$g(\mathcal{H}_q, i) = \prod_{l,\ (k,l)} \Big(\frac{1}{n_l} \sum_{\{j|m(j)=l\}} r_{j,i}\Big)^{1-I(l,k)} (r_{j,i})^{I(l,k)} \tag{4}$$

Algorithm 3 Neighbor-Expander

```
Require: The attributed graph $\mathcal{G}$, the query graph $\mathcal{H}_q$, one
    "touched" attribute value $k$ in $\mathcal{H}_q$, and the partially built
    subgraph $\mathcal{H}_t$.
    Output: One qualifying node $\langle i, k\rangle$ in $\mathcal{G}$
for each $\langle i, k\rangle$ in $\mathcal{G}$ do
    compute $g(\mathcal{H}_q, i)$ by equation (4)
end for
return: $i = \text{argmax}_{j\notin \mathcal{H}_t}\ r(\mathcal{H}_q, j)$
```

[2] In this paper, we always choose the attribute value with the highest degree in $\mathcal{H}_q$.

4.3 Bridge

Bridge takes as input two matching nodes i and j, and the attributed graph $\mathcal{G}$; and outputs a "best path" to connect i and j.

At first glance, we can use the "EXTRACT" algorithm in [19] or the display generation algorithm in [7]. However, the situation is different in our problem setting. First of all, as the matching subgraph $\mathcal{H}_t$ grows, some intermediate nodes might be already in the partially built $\mathcal{H}_t$, both "EXTRACT" [19] and display generation algorithm [7] will **favor** such kind of paths because of the total budget limitation on the size of the subgraph. However in our problem setting, we **forbid** such paths. Otherwise, $\mathcal{H}_t$ might not conform with the query graph $\mathcal{H}_q$ because of path overlap. What is more important, here, we only need to find one, rather than multiple "best" paths as in "EXTRACT" and display generation algorithm, which enable us to design a more efficient, Prim-like, algorithm. Formally, we define the "best path" between two matching nodes i and j as the one that maximizes the totally captured proximity score along the path over the total length of the path. Intuitively, a "best path" should contribute as much as possible for a particle to reach j from i when it does random walk with restart from node i.

The pseudo code of **Bridge** is given in Alg. (4). Note that in step 8, if the node v is already in the $\mathcal{H}_t$, we will block it.

Algorithm 4 Bridge

```
Require: The attributed graph $\mathcal{G}$, two matching nodes $i, j$,
    and the partially built subgraph $\mathcal{H}_t$.
    Output: One "best" path connecting node $i$ and $j$ in $\mathcal{G}$
 1: let $V$ be the total node set in $\mathcal{G}$: $V = \{1, 2, ..., n\}$,
 2: let $X = \{i\}$, $d(i) = r_{i,i}$, len$(i) = 1$, and Pre$(i) = i$
 3: for each node $u$ in $V$ do
 4:     $d(u) = 0$, len$(u) = 0$
 5: end for
 6: while $V$ is not empty do
 7:     $u = \text{argmax}_{\tilde{u} \in V} d(\tilde{u})$, move $u$ from $V$ to $X$
 8:     for each edge $(u, v)$ in $\mathcal{G}$, $v \in V$, and $v \notin \mathcal{H}_t$ do
 9:         if $d(v) < \frac{r_{i,v} + d(u)\text{len}(u)}{\text{len}(u)+1}$ then
10:             $d(v) = \frac{r_{i,v} + d(u)\text{len}(u)}{len(u)+1}$, len$(v)$=len$(u)$+1,Pre$(v)$=$u$
11:         end if
12:     end for
13: end while
14: Output the path from $i$ to $j$ by tracing back Pre$(j)$.
```

4.4 Efficiency Issues

In *G-Ray* we use random walk with restart. First of all, the size of the query graph $\mathcal{H}_q$ (usually less than 10 nodes) is much smaller than the attributed graphs, so the main time cost lies in the random walk with restart in $\mathcal{G}$. In this subsection, we first reduce the total number of random walks with restart by constructing an *augmented graph* (to be described next); and then we use a hybrid strategy to perform one only random walk with restart.

Based on equations (3, 4), we will have to do a lot of random walk with restart. For example, to one item in $g(\mathcal{H}_q, i)$ for a given node $\langle i, k \rangle$, we need n_l random walk with restart if edge (k, l) has been marked as "un-processed". Thus, totally we will need at most $(n_q + \prod_{l \in \mathcal{H}_q} n_l)$ random walk with restart, which might be very time consuming. However, based on the following Lemma, the number of random walks with restart can be largely reduced. We give the formal definition of the *augmented graph*, and then follow with an example (see Figure 3).

LEMMA 1. *Given an attributed graph $\mathcal{G} = \{\mathbf{W}, \mathbf{A}\}$, construct an augmented graph $\mathbf{W}'$ as equation (5). Let $r'_{j,i}(1 \leq i, j \leq n + m)$ be the* steady-state probability *that a particle will find itself at node i when it does random walk with restart from node j in the augmented $\mathbf{W}'$. Then the following equivalence holds:*

$$\frac{1}{n_l} \sum_{\{j|m(j)=l\}} r_{j,i} = \frac{1}{(1-c)} r'_{l+n,i}$$

$$\mathbf{W}' = \begin{pmatrix} \mathbf{W} & \mathbf{0} \\ \mathbf{A} & \mathbf{0} \end{pmatrix} \tag{5}$$

PROOF. Omitted for brevity □

In the augmented graph $\mathbf{W}'$, we refer to the newly added nodes as *attribute nodes*, and to the original nodes in $\mathbf{W}$ as *data nodes*. Intuitively, we put a directed edge from the attribute node to each of the data nodes having the corresponding attribute value. For example, Figure 3 is the augmented graph for the simplified social network in Figure 1. We introduce a new node for the attribute value CEO; and put a directed edge from this node to both nodes 12 and 13, respectively (For the other attribute values, we process similarly).

In order to measure the average proximity for a given node i wrt all the data nodes having attribute value l in $\mathcal{G}$, according to Lemma 1, we only need to do random walk with restart from the corresponding attribute node $(n+l)$ in the augmented graph $\mathbf{W}'$. Based on Lemma 1, it can be proved that we only need at most $2n_q$ random walk with restart on the augmented graph $\mathbf{W}'$.

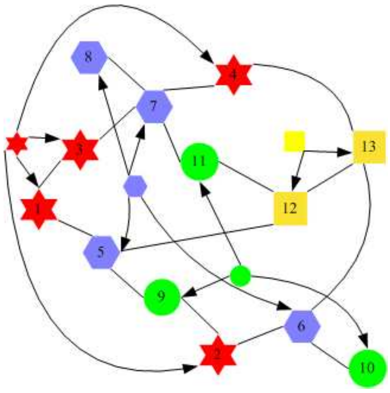


Figure 3: Augmented graph for the attributed graph in Figure 1. Small-size glyphs stand for "attribute" nodes, and have (directed) connections to the corresponding data nodes.

The most straightforward way to solve one random walk with restart is the iterative method [16], which is simple and accurate. However, it is slow for large graphs. In literatures, there are many fast/approximate solutions, e.g. BlockRank [12], Fingerprint based method [8], B_lin [20], etc. It should be pointed out that these methods are orthogonal to *G-Ray* – in principle, we can choose any of them. In this paper, we use a hybrid strategy: we use the B_Lin in [20] to generate a small fraction of the whole attributed graph $\mathcal{G}$ as the so-called candidate graph; and then run the whole algorithm on this candidate graph by the iterative method. As we will show in the next section, this strategy will largely reduce the response time (usually 1 order of magnitude faster).

5. EXPERIMENTAL EVALUATION

We present experiments to answer the following questions:

- *effectiveness* of our goodness function $g()$: that is, whether the matching graphs agree with our intuition.

- *speed* and *scalability*: how does *G-Ray* scale up for large graphs.

We present the results next, after we describe our experimental setup.

5.1 Experimental Setup

5.1.1 Data

We use the DBLP data set[3] to construct the attributed graph, where the nodes are authors and the attribute is the conference name (and year, e.g., 'KDD-2001'). The node-to-node matrix $\mathbf{W}$ is constructed from the authorship ($w_{i,j}$ is the number of the co-authored paper between author i and j); the node-to-attribute matrix $\mathbf{A}$ is constructed from author-conference relationship ($a_{i,j} = 1$ if the author i has ever published in the conference j, 0 otherwise). In total, there are n=356,364 nodes; E=1,905,970 edges, and m=12,920 attribute values in the graph.

5.1.2 Parameter Settings

The selection of the appropriate size for the candidate graph is a trade-off between the response time and the quality/goodness of the resulting subgraphs. We perform a parametric study as follows: for a given size of candidate graph, we issue a 4-node query and return the top-5 subgraphs. We test different types of queries (namely, line-query, loop-query, and star-query). For each type of query, the experiment is run multiple times.

Figure 4 shows the mean log quality/goodness vs. the average response time per subgraph. There is a plateau in Figure 4 at (log) goodness = -30, starting at 3 seconds of average response time). At this point, the size of the candidate graph is 1% of the whole graph. Thus this is the ratio that we use in the remaining experiments.

There are two parameters left, the fly-out probability c of random walk with restart, and the number of iterations for the iterative method. In all the experiments, c is set to 0.1 and the number of iterations is set to 50 since no performance improvement is observed with more iterations.

5.2 Effectiveness

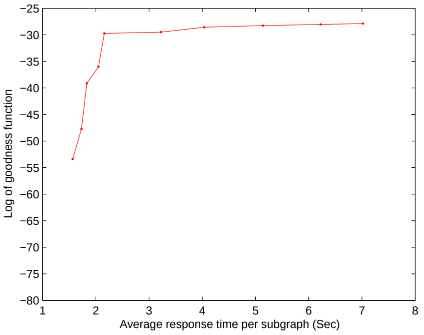


Figure 4: Quality vs. response time. Notice the plateau, starting at about 3 seconds.

The question is how effective our proposed goodness function $g()$ is, and whether the subgraphs that *G-Ray* retrieves would agree with the intuition of a domain expert.

Figures 5.a-f show three queries (star, line, loop) and the resulting subgraphs retrieved from the DBLP graph. In all the cases, the results make sense.

Let us analyze the 'star' query first, which requests a star-shape group of co-authors, with one author from each of PODS, IAT ('Intelligent Agent technology') and ISBMS ('International Symposium on Biomedical Simulation'). We see that Philip Yu is in the center, with the rest of the matching nodes being well-known domain experts (Haixun Wang and Mark Zhang for Agents); the connection to biomedical simulation is strained, requiring an interception by Bing Liu.

For the line query ('find a chain of co-authors, from STOC to SIGMOD to ICML to ISBMS'), again *G-Ray* retrieves well established researchers from theory (Charikar), databases (Garcia-Molina), machine learning (Fayyad). Again, the connection to biomedical simulation is strained, requiring 3 intermediate nodes (in white/unshaded).

The loop query (KDD, RECOMB, INFOCOMM, and ICML) is also very interesting. There is a gap between KDD96 and RECOMB00 (biology); there is a surprising, direct link between biomedical and computer networks (Karp-Shenker); and there is a long path from INFOCOMM00 to ICML93 (probably due to both the chronological difference, as well as the lack of interaction between the research communities).

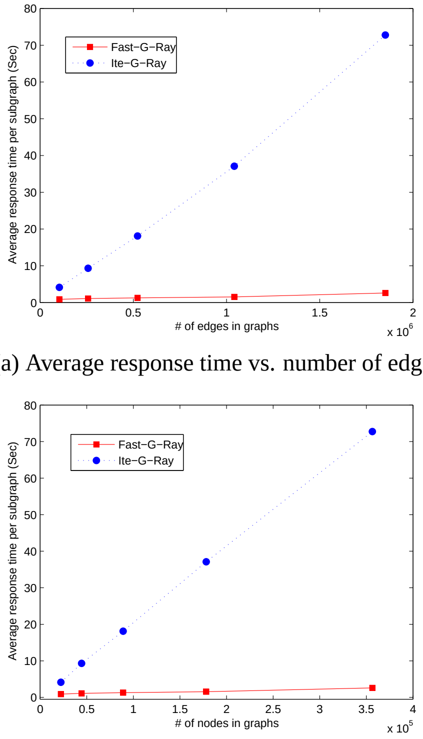
5.3 Efficiency

We use different sizes of subsets of the whole DBLP data set to test how *G-Ray* scales with the size of the graph. For each subset, we randomly generate a 4-node query of different types (star-query, line-query, and loop-query) and return the top-5 subgraphs. For each type of query, we run the experiment multiple times and report the average response time. We compared two strategies to perform random walk with restart *G-Ray*: (1) using an iterative method on the whole subset (Ite-*G-Ray*) and (2) using a hybrid strategy as described in Section 4.4 (Fast-*G-Ray*).

The average response time per subgraph vs. the number of nodes/edges is presented in Figure 6. It can be seen that in both cases, *G-Ray* scales linearly with the size of the graphs. More importantly, Fast-*G-Ray* scales linearly with a much smaller

[3] `http://www.informatik.uni-trier.de/~ley/db/`

slope. For example, on the full size of graph (356K nodes and 1.9M edges), the average response time per subgraph is *3 seconds*, while it takes more than 1 minute for Ite-*G-Ray*.



(a) Average response time vs. number of edges

(b) Average response time vs. number of nodes

Figure 6: Scalability of *G-Ray*. Time versus data graph size. Both versions of *G-Ray* scale linearly, with Fast-*G-Ray* (bottom) having significantly lower slope.

6. RELATED WORK

Graph matching algorithms vary widely due to differences in the specific problems they address. *G-Ray* is a fast approximate algorithm for inexact pattern matching in large, (categorically) attributed graphs. *G-Ray* extends the ideas of connection subgraphs [7] and centerpiece graphs [19, 20] and applies them to pattern matching in attributed graphs. This work is also related to the idea of network proximity, which builds on connection subgraphs as well [13].

While there has been a large amount of work on graph matching over the past 30 years, much of it is not directly applicable to our problem setting. Many graph matching techniques focus strictly on matching graph structure and do not utilize attributes. Other work focuses on exact matching, but cannot handle inexact matching. Still other methods focus on matching against a database of many small graphs (i.e., the graph-transaction setting) instead of a single large graph (i.e., the single-graph setting). The single-graph setting is more general and algorithms developed for single graphs can be readily applied to the graph-transaction setting, although the converse is not true [15]. For additional background on graph matching algorithms, we refer the reader to a recent survey by Gallagher [9].

There has been significant work on inexact graph matching [18, 21, 10, 22, 5, 1], on matching attributed graphs [21, 18, 10, 3, 22, 5], and on matching in the single-graph setting [3, 22, 5, 1]. However, there are relatively few algorithms that combine the three to tackle inexact matching in large, attributed graphs [6, 22, 5, 1]. Furthermore, while these algorithms employ various optimizations to mitigate the computational complexity of the problem, they all exhibit super-linear complexity in the worst case. Unfortunately, it is also difficult to determine the performance characteristics of these algorithms due to a lack of reported results and complexity analysis.

In addition to the graph matching work described above, there is related work of interest in the database and data mining literature. Our work focuses on finding instances of user-specified patterns in graphs. Related problems include discovery of frequent or interesting patterns (i.e., graph mining) and inexact querying of databases.

Yan, Yu, and Han propose efficient methods for indexing and mining graph databases based on the occurrence of frequent substructures [23, 24]. Jin et al. use the concept of a topological minor to quickly discover frequent large-scale patterns [11]. As with many of the graph matching techniques described above, these mining algorithms are designed for graph-transactional databases (e.g., collections of biological or chemical structures) and are not readily applicable to the single-graph setting. Cook and Holder [6] and Kuramochi and Karypis [15] propose algorithms for graph mining in the single-graph setting. The empirical evaluation by the latter shows that their method outperforms that of Cook and Holder in terms of runtime on a number of real data sets. Pei et al. [17] take on a somewhat different graph mining task. Their goal is to discover quasi-clique patterns across multiple related graph data sets (e.g., groups of customers with similar behavior across markets). We refer the reader to Chakrabarti's book [4] on web mining for a more information on web and graph mining techniques.

We also find related work in the area of inexact querying of relational databases. Koudas et al. [14] propose a method for relaxing relational database queries to accommodate near, but inexact matches. However, this work does not support inexact structural matching. The method will relax attribute value conditions and join conditions, but there is no flexibility in terms of what relations are involved in the joins. The BANKS system proposed by Bhalotia et al. [2] enables a user to issue keyword-based queries to a relational database without any knowledge of the underlying database schema. BANKS models database tuples as nodes in a graph, but is restricted to return tree-structured results. *G-Ray* imposes no such restriction. In addition, BANKS assesses relevance of results based on the proximity of matching nodes and an information retrieval inspired weighting scheme. In our method, results are ranked according to our goodness function (which is based on proximity of nodes in a graph).

7. CONCLUSION

We have addressed the problem of finding best-effort subgraph patterns in attributed graphs. The typical query is, say, '*find a potential money laundering ring, consisting of alternating nodes of businessmen and bankers*'. To the best of our knowledge, this is the first method that returns best-effort results, even when the exact pattern does not exist in the data set. The second major characteristic of our method is that it scales very well with the database size: our experiments show that the wall-clock time grows near-linearly with the size of the graph.

We also report experiments on the DBLP dataset (356K nodes, 1.9M edges), where the results agree with intuition, and the wall-clock time is about 3-5 seconds, on a commodity PC.

Future work includes extension to handle attributes on the edges.

8. ACKNOWLEDGEMENT

This work was supported by and performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48. UCRL-CONF-XXXXXX.

9. REFERENCES

[1] B. Aleman-Meza, C. Halaschek-Wiener, S. Sahoo, A. Sheth, and I. Arpinar. *Lecture Notes in Computer Science*, volume 3495, chapter Template Based Semantic Similarity for Security Applications, pages 621–622. Springer, 2005.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, pages 431–440, 2002.

[3] H. Blau, N. Immerman, and D. Jensen. A visual language for querying and updating graphs. Technical Report 2002-037, Department of Computer Science, University of Massacheusetts, Amherst, 2002.

[4] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002.

[5] T. Coffman, S. Greenblatt, and S. Marcus. Graph-based technologies for intelligence analysis. *Communications of the ACM, Special Issue on Emerging Technologies for Homeland Security*, 47(3):45–47, 2004.

[6] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research (JAIR)*, 1:231–255, 1994.

[7] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD '04: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 118127, 2004.

[8] D. Fogaras and B. Racz. Towards scaling fully personalized pagerank. In *Proc. WAW*, pages 105–117, 2004.

[9] B. Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. In *AAAI FS '06: Papers from the 2006 AAAI Fall Symposium on Capturing and Using Patterns for Evidence Detection*, pages 45–53, 2006.

[10] N. Guarino, C. Masolo, and G. Vetere. Ontoseek: Content-based access to the web. *IEEE Intelligent Systems*, 14(3):70–80, 1999.

[11] R. Jin, C. Wang, D. Polshakov, S. Parthasarathy, and G. Agrawal. Discovering frequent topological structures from graph datasets. In *KDD '05: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 606–611, 2005.

[12] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing pagerank. In *Stanford University Technical Report*, 2003.

[13] Y. Koren, S. North, and C. Volinsky. Measuring and extracting proximity in networks. In *KDD '06: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 245–255, 2006.

[14] N. Koudas, C. Li, A. Tung, and R. Vernica. Relaxing join and selection queries. In *VLDB '06: Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 199–210, 2006.

[15] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery*, 11(3):243–271, 2005.

[16] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.

[17] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *KDD '05: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2005.

[18] L. Shapiro and R. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:504–519, 1981.

[19] H. Tong and C. Faloutsos. Center-piece subgraphs: Problem definition and fast solutions. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 404–413, 2006.

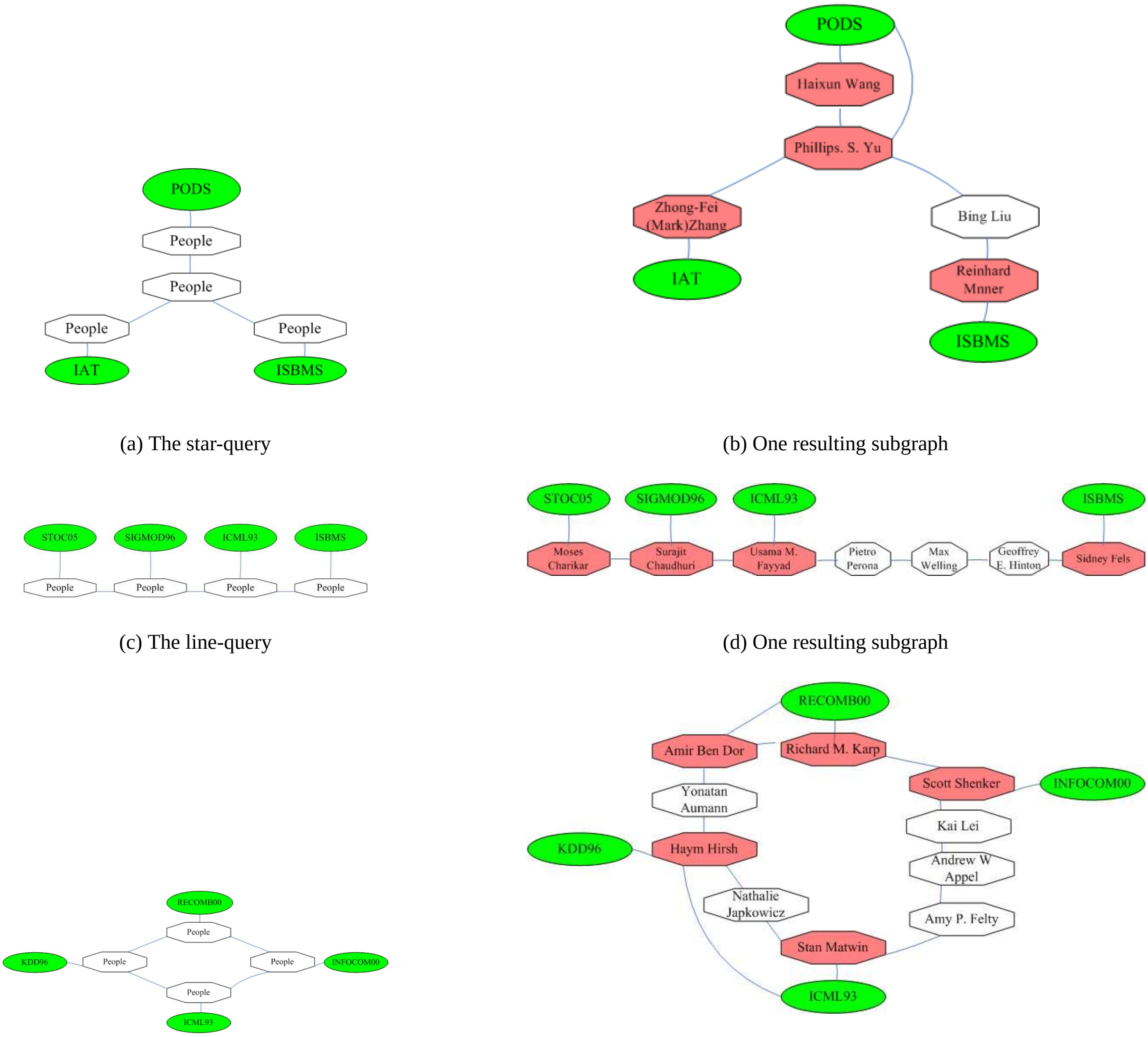
[20] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast random walk with restart and its applications. In *ICDM '06: Proceedings of the 6th IEEE International Conference on Data Mining*, pages 613–622, 2006.

[21] W.-H. Tsai and K.-S. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Transactions on Systems, Man and Cybernetics*, 9:757–768, 1979.

[22] M. Wolverton, P. Berry, I. W. Harrison, J. D. Lowrance, D. Morley, A. C. Rodriguez, E. H. Ruspini, and J. Thoméré. Law: A workbench for approximate pattern matching in relational data. In *IAAI '03: Proceedings of the Fifteenth Conference on Innovative Applications of Artificial Intelligence*, pages 143–150, 2003.

[23] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM '02: Proceedings of the 2nd IEEE International Conference on Data Mining*, pages 721–724, 2002.

[24] X. Yan, P. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *ICDM '04: Proceedings of the 4th International Conference on Data Mining*, pages 335–346, 2004.



(a) The star-query

(b) One resulting subgraph

(c) The line-query

(d) One resulting subgraph

(e) The loop-query

(f) One resulting subgraph

Figure 5: Some typical queries on DBLP data set and some of their results.